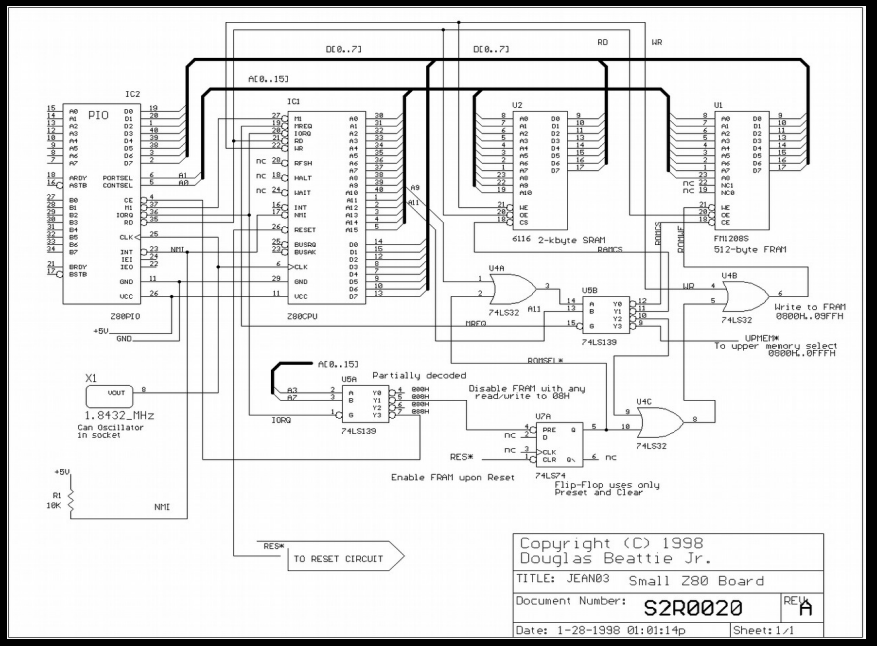
Desarrollo de una arquitectura en Radare2

Joan Soriano

Introducción



Índice

→ ¿Qué es Radare?

→ Tipos de plugins

→ Instalación de plugins

→ Referencias

ALERT

Turra Time

Radare2

¿Qué es?

Radare2 es un framework multiplataforma de reversing

- Comunidad

- Do you want it? Do it

- Open Source

Comparación

Analysis	r2	IDA	Hopper
Call/syscall recognition	yes	yes	yes
Cross-references	yes	yes	yes
DWARF support	yes	yes	yes
dSym support	yes	yes	yes
Signature recognition	yes	yes	yes
Custom structures definition	yes	yes	no
Type support	yes	yes	yes
Diffing	yes	plugin	no

Debugger	r2	IDA	Hopper
Breakpoints	yes	yes	yes
Process attaching	yes	yes	yes
Remote debugging	yes	yes	yes
Tracing	yes	yes	no
Emulation	yes	plugin	no

Architectures	r2	IDA	Hopper
ARM	yes	yes	yes
CSR	yes	no	no
Java/Dalvik	yes	yes	no
TMS320C55x+	yes	no	no
v850	yes	yes	no
x86 and x64	yes	yes	yes

Prices and support	r2	IDA	Hopper
Free (as in beer)	yes	450-2700€	65-125€
Free (as in freedom)	yes	no	no
Open bugtracker	yes	no	yes
Professional customers support	no	yes	yes

Comparación

“Maldito pagano comunista, como no digas en voz alta que adoras a la Virgen María, voy a hacer papillas con tus tripas”

Estructura de Radare2

¿Cómo funciona Radare2?

Radare2 está desarrollado de manera que divide la lógica de la CPU en diferentes módulos.

Para dar un soporte completo para una arquitectura específica, deben desarrollarse más de un plugin.

¿Cómo funciona Radare2?

Los plugins pueden estar compilados estáticamente o dinámicamente, es decir, que la arquitectura esté dentro de las librerías del núcleo de Radare2 o distribuidas como una librería compartida

¿Cómo funciona Radare2?

- Según la wiki de Radare2, una arquitectura tiene soporte cuando tiene implementados los siguientes plugins:

    - r_asm
    - r_anal
    - r_reg
    - r_syscall
    - r_debug

¿Y esto para qué me sirve?

¿Y esto, para qué sirve?

Colaboradores habituales de Radare

¿Cómo funciona Radare2?

“Tengo una teoría.
Hay dos clases de niños, los que quieren ser astrónomos, y los que quieren ser astronautas”

¿Por dónde empezamos?

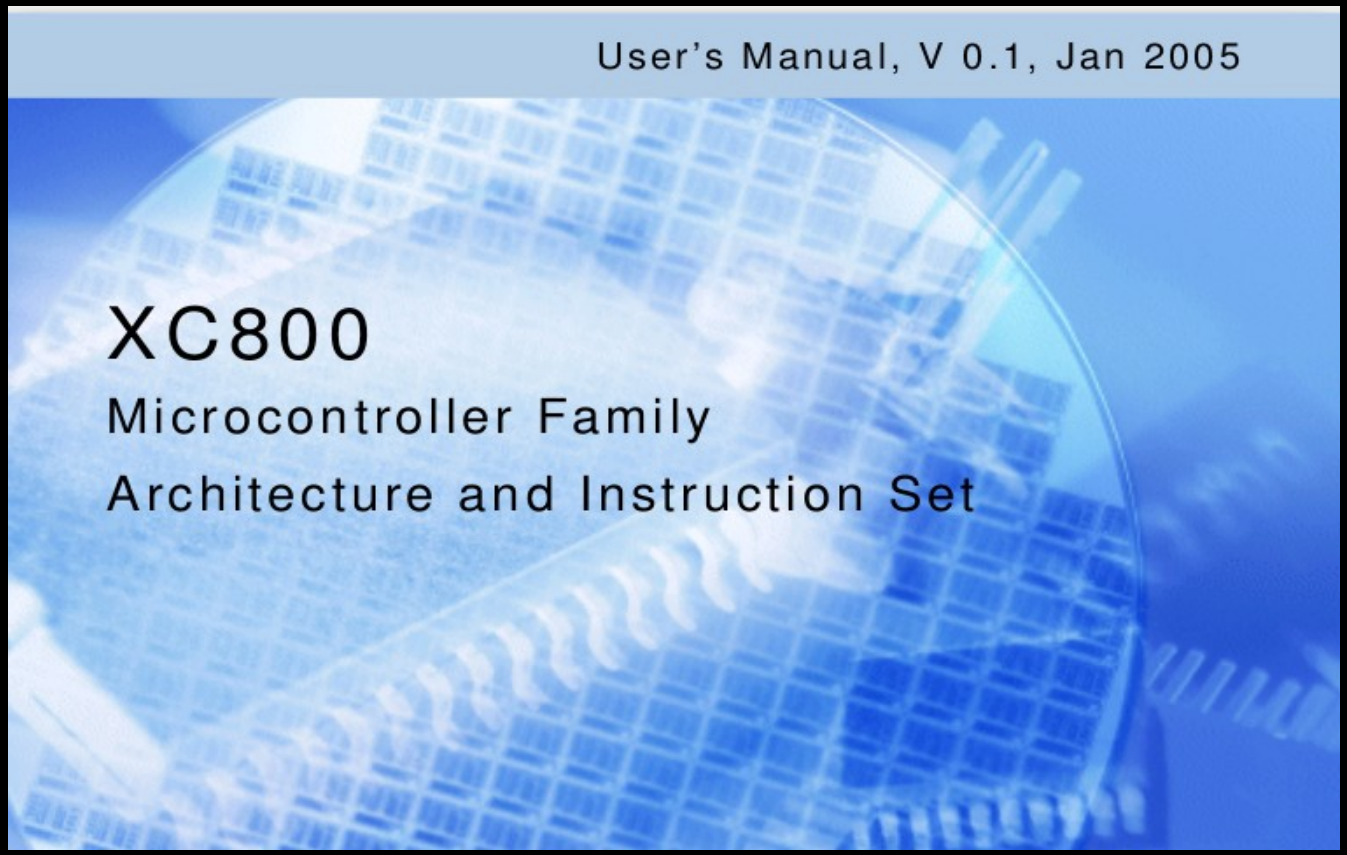
¿Por dónde empezamos?

- Datasheet

- Arquitecturas implementadas con anterioridad

¿Por dónde empezamos?

- Datasheet



¿Por dónde empezamos?

"Necesito tu ropa, tus botas y tu motocicleta"

¿Por dónde empezamos?

- Arquitecturas implementadas con anterioridad

  xc 800 similar a:

  - Z80
  - 8051
  - i8080

Paso 1

Struct plugin

Definir qué tipo de plugin es:

```
struct r_lib_struct_t radare_plugin = {
        .type = R_LIB_TYPE_BIN,
        .data = &r_bin_plugin_xc800,
        .version = R2_VERSION
};
```

¿Por dónde empezamos?

Tipos de plugins:

- io
- debugger
- assembler
- analysis
- parsers
- bins
- bin extractors
- syscall
- fastcall
- cryptography
- Rcore commands
- r_egg plugin
- r_fs plugin

r_bin

r_bin

bin_nin3ds.c

```
struct r_bin_plugin_t r_bin_plugin_nin3ds = {
        .name = "nin3ds",
        .desc = "Nintendo 3DS FIRM format r_bin plugin",
        .license = "LGPL3",
        .load = &load,
        .load_bytes = &load_bytes,
        .destroy = &destroy,
        .check = &check,
        .check_bytes = &check_bytes,
        .entries = &entries,
        .sections = &sections,
        .info = &info,
};
```

r_bin

- name: Nombre del plugin
- desc: Descripción del plugin
- load:
- load_bytes:
- destroy: Vaciado de buffer
- check:
- check_bytes:
- entries: Punto de entrada de binarios
- sections: Secciones de la memoria
- symbols: Interrupciones
- mem: Diferenciación de tipos de memoria
- info

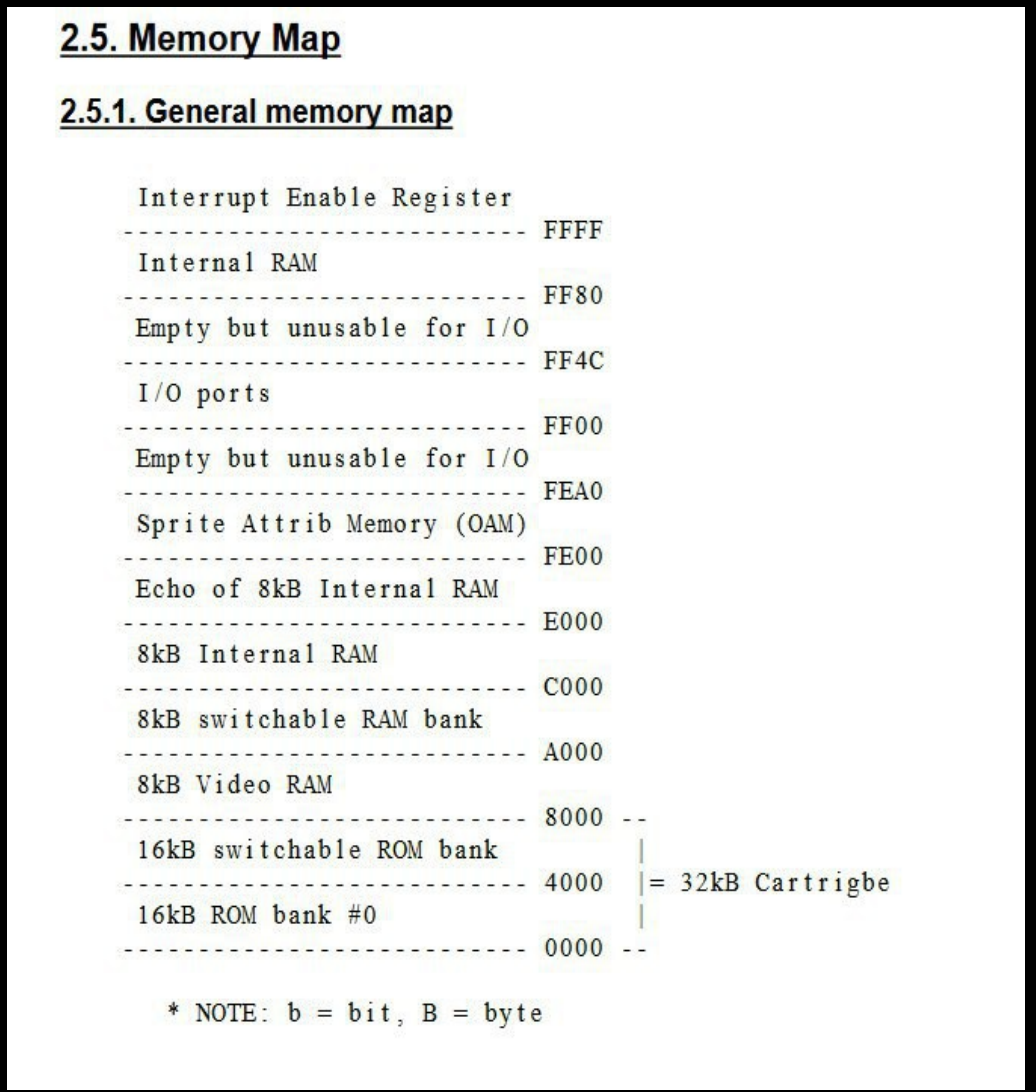
r_bin

```
- name: “r_bin_xc800”
- desc: “xc800 r_bin plugin”
```

r_bin

2.5. Memory Map

2.5.1. General memory map



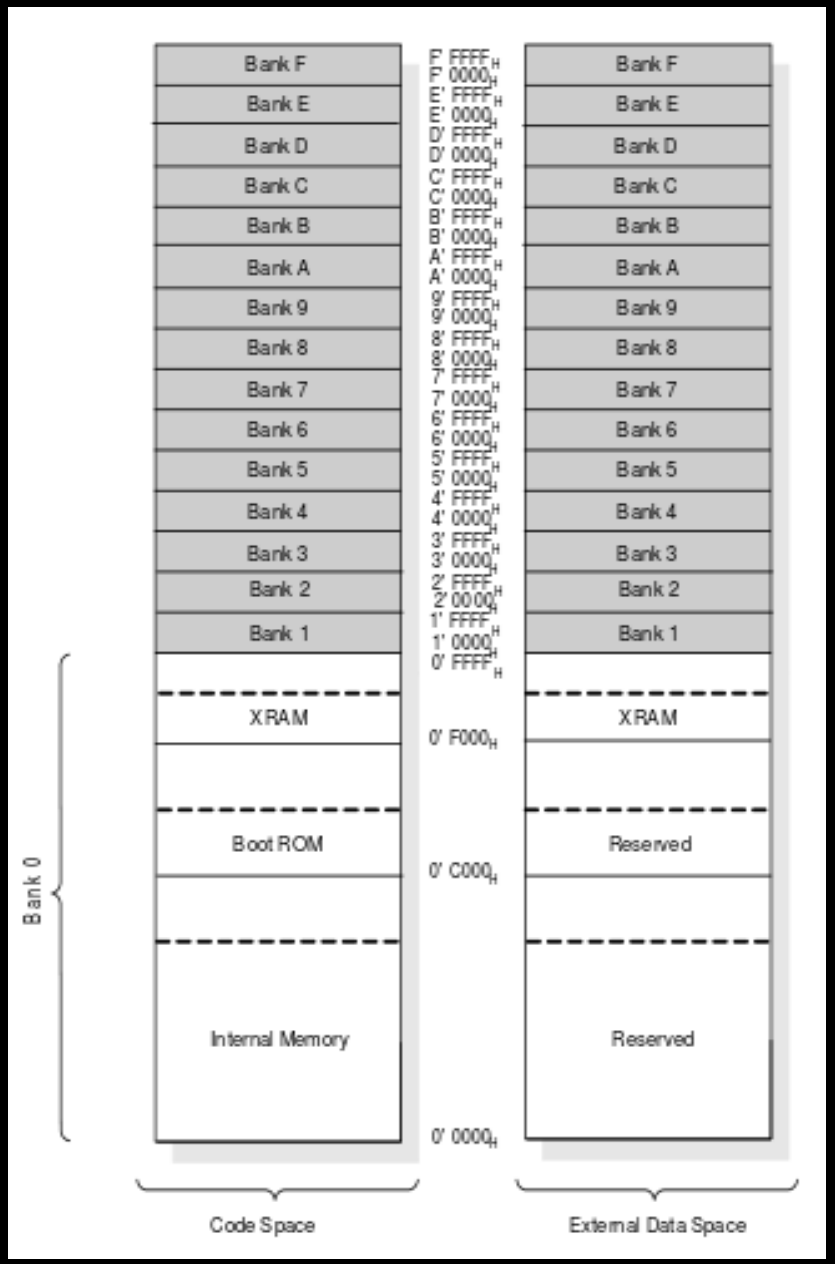
* NOTE: b = bit, B = byte

sections

Sections en Z80

```
        for (i = 1; i < bank; i++) {
                rombank[i] = R_NEW0 (RBinSection);
                sprintf (rombank[i]->name,"rombank%02x",i);
                rombank[i]->paddr = i*0x4000;
                rombank[i]->vaddr = i*0x10000-0xc000;
                rombank[i]->size = rombank[i]->vsize = 0x4000;
                rombank[i]->srwx = r_str_rwx ("mrx");
                rombank[i]->add = true;
                r_list_append (ret,rombank[i]);
        }
        return ret;
}
```

sections



Pág. 8 “XC800 User's Manual”

sections

```
        bbank[0] = R_NEW0 (RBinSection);
        strncpy (bbank[0]->name, "bank0", R_BIN_SIZEOF_STRINGS);
        bbank[0]->paddr = 0;
        bbank[0]->size = 0xFFFF;
        bbank[0]->vsize = 0xFFFF;
        bbank[0]->vaddr = 0;
        bbank[0]->srwx = r_str_rwx ("mrx");
        bbank[0]->add = true;

        r_list_append (ret, bbank[0]);

        for (i = 1; i < 16; i++) {
                bbank[i] = R_NEW0 (RBinSection);
                sprintf (bbank[i]->name,"bank%01x",i);
                bbank[i]->paddr = i*0x10000;
                bbank[i]->vaddr = i*0x10000;
                bbank[i]->size = bbank[i]->vsize = 0xFFFF;
                bbank[i]->srwx = r_str_rwx ("mrx");
                bbank[i]->add = true;
                r_list_append (ret,bbank[i]);
        }
        return ret;
}
```

sections

```
[0x00000000]> iS

[Sections]
idx=00 vaddr=0x00000000 paddr=0x00000000 sz=65535 vsz=65535 perm=m-r-x name=bank0
idx=01 vaddr=0x00010000 paddr=0x00010000 sz=65535 vsz=65535 perm=m-r-x name=bank1
idx=02 vaddr=0x00020000 paddr=0x00020000 sz=65535 vsz=65535 perm=m-r-x name=bank2
idx=03 vaddr=0x00030000 paddr=0x00030000 sz=65535 vsz=65535 perm=m-r-x name=bank3
idx=04 vaddr=0x00040000 paddr=0x00040000 sz=65535 vsz=65535 perm=m-r-x name=bank4
idx=05 vaddr=0x00050000 paddr=0x00050000 sz=65535 vsz=65535 perm=m-r-x name=bank5
idx=06 vaddr=0x00060000 paddr=0x00060000 sz=65535 vsz=65535 perm=m-r-x name=bank6
idx=07 vaddr=0x00070000 paddr=0x00070000 sz=65535 vsz=65535 perm=m-r-x name=bank7
idx=08 vaddr=0x00080000 paddr=0x00080000 sz=65535 vsz=65535 perm=m-r-x name=bank8
idx=09 vaddr=0x00090000 paddr=0x00090000 sz=65535 vsz=65535 perm=m-r-x name=bank9
idx=10 vaddr=0x000a0000 paddr=0x000a0000 sz=65535 vsz=65535 perm=m-r-x name=banka
idx=11 vaddr=0x000b0000 paddr=0x000b0000 sz=65535 vsz=65535 perm=m-r-x name=bankb
idx=12 vaddr=0x000c0000 paddr=0x000c0000 sz=65535 vsz=65535 perm=m-r-x name=bankc
idx=13 vaddr=0x000d0000 paddr=0x000d0000 sz=65535 vsz=65535 perm=m-r-x name=bankd
idx=14 vaddr=0x000e0000 paddr=0x000e0000 sz=65535 vsz=65535 perm=m-r-x name=banke
idx=15 vaddr=0x000f0000 paddr=0x000f0000 sz=65535 vsz=65535 perm=m-r-x name=bankf

16 sections
```

symbols

2.12.2. Interrupt Descriptions

The following interrupts only occur if they have been enabled in the Interrupt Enable register ($FFFF) and if the interrupts have actually been enabled using the EI instruction.

1. V-Blank

The V-Blank interrupt occurs ~59.7 times a second on a regular GB and ~61.1 times a second on a Super GB (SGB). This interrupt occurs at the beginning of the V-Blank period. During this period video hardware is not using video ram so it may be freely accessed. This period lasts approximately 1.1 ms.

2. LCDC Status

There are various reasons for this interrupt to occur as described by the STAT register ($FF40). One very popular reason is to indicate to the user when the video hardware is about to redraw a given LCD line. This can be useful for dynamically controlling the SCX/SCY registers ($FF43/$FF42) to perform special video effects.

3. Timer Overflow

This interrupt occurs when the TIMA register ($FF05) changes from $FF to $00.

4. Serial Transfer Completion

This interrupt occurs when a serial transfer has completed on the game link port.

symbols

Symbols en Z80

```
ptr[8]->name = strdup ("Interrupt_Vblank");
ptr[8]->paddr = ptr[8]->vaddr = 64;
ptr[8]->size = 1;
ptr[8]->ordinal = 8;
r_list_append (ret, ptr[8]);

if (!(ptr[9] = R_NEW0 (RBinSymbol)))
        return ret;

ptr[9]->name = strdup ("Interrupt_LCDC-Status");
ptr[9]->paddr = ptr[9]->vaddr = 72;
ptr[9]->size = 1;
ptr[9]->ordinal = 9;
r_list_append (ret, ptr[9]);

if (!(ptr[10] = R_NEW0 (RBinSymbol)))
        return ret;

ptr[10]->name = strdup ("Interrupt_Timer-Overflow");
ptr[10]->paddr = ptr[10]->vaddr = 80;
ptr[10]->size = 1;
ptr[10]->ordinal = 10;
r_list_append (ret, ptr[10]);
```

symbols

Table 2-2 Interrupt Vector Addresses

Interrupt Source	Vector Address	Interrupt Sources
XINTR0	0003_H	External Interrupt 0
XINTR1	$000B_H$	Timer 0
XINTR2	0013_H	External Interrupt 1
XINTR3	$001B_H$	Timer 1
XINTR4	0023_H	UART
XINTR5	$002B_H$	Extended Interrupt 5 (Timer 2)
XINTR6	0033_H	Extended Interrupt 6
XINTR7	$003B_H$	Extended Interrupt 7
XINTR8	0043_H	Extended Interrupt 8
XINTR9	$004B_H$	Extended Interrupt 9
XINTR10	0053_H	Extended Interrupt 10
XINTR11	$005B_H$	Extended Interrupt 11
XINTR12	0063_H	Extended Interrupt 12
XINTR13	$006B_H$	Extended Interrupt 13
NMI	0073_H	Non-maskable Interrupt

Pág. 37 “XC800 User's Manual”

symbols

```
static RList* symbols(RBinFile *arch) {
        RList *ret = NULL;
        RBinSymbol *ptr[15];
        int i;
        if (!(ret = r_list_new()))
                return NULL;
        ret->free = free;
```

```
if (!(ptr[0] = R_NEW0 (RBinSymbol)))
        return ret;

ptr[0]->name = strdup ("External Interrupt 0");
ptr[0]->paddr = ptr[0]->vaddr = 3;
ptr[0]->size = 1;
ptr[0]->ordinal = 0;
r_list_append (ret, ptr[0]);

if (!(ptr[1] = R_NEW0 (RBinSymbol)))
        return ret;

ptr[1]->name = strdup ("Timer 0");
ptr[1]->paddr = ptr[1]->vaddr = 11;
ptr[1]->size = 1;
ptr[1]->ordinal = 1;
r_list_append (ret, ptr[1]);

if (!(ptr[2] = R_NEW0 (RBinSymbol)))
        return ret;

ptr[2]->name = strdup ("External Interrupt 1");
ptr[2]->paddr = ptr[2]->vaddr = 19;
ptr[2]->size = 1;
ptr[2]->ordinal = 2;
r_list_append (ret, ptr[2]);
```

symbols

```
[0x00000100]> is
[Symbols]
vaddr=0x00000003 paddr=0x00000003 ord=000 fwd= sz=1 bind= type= name=External Interrupt 0
vaddr=0x0000000b paddr=0x0000000b ord=001 fwd= sz=1 bind= type= name=Timer 0
vaddr=0x00000013 paddr=0x00000013 ord=002 fwd= sz=1 bind= type= name=External Interrupt 1
```

info

Info en Z80

```
static RBinInfo* info(RBinFile *arch) {
        ut8 rom_header[76];
        RBinInfo *ret = R_NEW0 (RBinInfo);
        if (!ret || !arch || !arch->buf) {
                free (ret);
                return NULL;
        }
        r_buf_read_at (arch->buf, 0x104, rom_header, 76);
        ret->file = calloc (1, 17);
        strncpy (ret->file, (const char*)&rom_header[48], 16);
        ret->type = malloc (128);
        ret->type[0] = 0;
        gb_get_gbtype (ret->type, rom_header[66], rom_header[63]);
        gb_add_cardtype (ret->type, rom_header[67]); // XXX
        ret->machine = strdup ("Gameboy");
        ret->os = strdup ("any");
        ret->arch = strdup ("gb");
        ret->has_va = true;
        ret->bits = 16;
        ret->big_endian = 0;
        ret->dbg_info = 0;
        return ret;
}
```

0134-0142 Title of the game in UPPER CASE ASCII. If it is less than 16 characters then the remaining bytes are filled with 00's.

0143 $80 = Color GB, $00 or other = not Color GB

0144 Ascii hex digit, high nibble of licensee code (new).

0145 Ascii hex digit, low nibble of licensee code (new). (These are normally $00 if [$014B] <> $33.)

0146 GB/SGB Indicator (00 = GameBoy, 03 = Super GameBoy functions)
(Super GameBoy functions won't work if <> $03.)

0147 Cartridge type:

0-ROM ONLY	12-ROM+MBC3+RAM
1-ROM+MBC1	13-ROM+MBC3+RAM+BATT
2-ROM+MBC1+RAM	19-ROM+MBC5

info

```
[0x00000100]> px
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00000100  00c3 5001 ceed 6666 cc0d 000b 0373 0083  ..P...ff....s..
0x00000110  000c 000d 0008 111f 8889 000e dccc 6ee6  ..............n.
0x00000120  dddd d999 bbbb 6763 6e0e eccc dddc 999f  ......gcn.......
0x00000130  bbb9 333e 504f 4b45 4d4f 4e20 5245 4400  ..3>POKEMON RED.
0x00000140  0000 0000 3031 0313 0503 0133 0020 91e6  ....01.....3. ..
0x00000150  fe11 2803 af18 023e 00ea 1acf c354 1f3e  ..(....>.....T.>
0x00000160  200e 00e0 00f0 00f0 00f0 00f0 00f0 00f0   ...............
0x00000170  002f e60f cb37 473e 10e0 00f0 00f0 00f0  ./...7G>........
```

0134-0142 Title of the game in UPPER CASE ASCII. If it is less than 16 characters then the remaining bytes are filled with 00's.

0143 $80 = Color GB, $00 or other = not Color GB

0144 Ascii hex digit, high nibble of licensee code (new).

0145 Ascii hex digit, low nibble of licensee code (new). (These are normally $00 if [$014B] <> $33.)

0146 GB/SGB Indicator (00 = GameBoy, 03 = Super GameBoy functions) (Super GameBoy functions won't work if <> $03.)

0147 Cartridge type:

0-ROM ONLY	12-ROM+MBC3+RAM
1-ROM+MBC1	13-ROM+MBC3+RAM+BATT
2-ROM+MBC1+RAM	19-ROM+MBC5

```
[0x00000100]> ixx
type      SuperGameboy-Rom
card     ROM+MBC3+RAM+BAT
file      POKEMON RED
fd        6
size      0x100000
iorw      false
blksz     0x0
mode      -r--
block     0x100
format    ningb
havecode  true
pic       false
canary    false
nx        false
crypto    false
va        true
arch      gb
bits      16
machine   Gameboy
os        any
```

r_asm

r_asm

asm_8051.c

```
RAsmPlugin r_asm_plugin_8051 = {
        .name = "8051",
        .arch = "8051",
        .bits = 8,
        .endian = R_SYS_ENDIAN_NONE,
        .desc = "8051 Intel CPU",
        .disassemble = &disassemble,
        .assemble = NULL,
        .license = "PD"
};
```

r_asm

```
static int disassemble(RAsm *a, RAsmOp *op, const ut8 *buf, int len) {
        char *tmp = NULL;

        xc800_op o = xc800_decode (buf, len);
        memset(op->buf_asm, 0, sizeof (op->buf_asm));
        if (!o.name) return 0; // invalid instruction
        tmp = xc800_disasm (o, a->pc, op->buf_asm, sizeof (op->buf_asm));
        if (tmp) {
                if (strlen(tmp) < sizeof (op->buf_asm)) {
                        strncpy (op->buf_asm, tmp, strlen (tmp));
                } else {
                        eprintf ("Too big opcode!\n");
                        free (tmp);
                        op->size = -1;
                        return -1;
                }
                free (tmp);
        }
        if (!*op->buf_asm) {
                op->size = 1;
                return -1;
        }
        return (op->size = o.length);
}
```

r_asm

Mnemonic	Description	Hex Code	Bytes	Cycles
	ARITHMETIC			
ADD A,Rn	Add register to A	28-2F	1	1
ADD A,direct	Add direct byte to A	25	2	1
ADD A,@Ri	Add indirect memory to A	26-27	1	1
ADD A,#data	Add immediate to A	24	2	1
ADDC A,Rn	Add register to A with carry	38-3F	1	1
ADDC A,direct	Add direct byte to A with carry	35	2	1
ADDC A,@Ri	Add indirect memory to A with carry	36-37	1	1
ADDC A,#data	Add immediate to A with carry	34	2	1
SUBB A,Rn	Subtract register from A with borrow	98-9F	1	1
SUBB A,direct	Subtract direct byte from A with borrow	95	2	1
SUBB A,@Ri	Subtract indirect memory from A with borrow	96-97	1	1
SUBB A,#data	Subtract immediate from A with borrow	94	2	1
INC A	Increment A	04	1	1
INC Rn	Increment register	08-0F	1	1

r_asm

Mnemonic	Description	Hex Code	Bytes	Cycles
INC DPTR	Increment data pointer	A3	1	2
MUL AB	Multiply A by B	A4	1	4
DIV AB	Divide A by B	84	1	4
DA A	Decimal Adjust A	D4	1	1
LOGICAL				
ANL A,Rn	AND register to A	58-5F	1	1
ANL A,direct	AND direct byte to A	55	2	1
ANL A,@Ri	AND indirect memory to A	56-57	1	1
ANL A,#data	AND immediate to A	54	2	1
ANL direct,A	AND A to direct byte	52	2	1
ANL direct,#data	AND immediate to direct byte	53	3	2
ORL A,Rn	OR register to A	48-4F	1	1
ORL A,direct	OR direct byte to A	45	2	1
ORL A,@Ri	OR indirect memory to A	46-47	1	1
ORL A,#data	OR immediate to A	44	2	1
ORL direct,A	OR A to direct byte	42	2	1
ORL direct,#data	OR immediate to direct byte	43	3	2
XRL A,Rn	Exclusive-OR register to A	68-6F	1	1
XRL A,direct	Exclusive-OR direct byte to A	65	2	1
XRL A,@Ri	Exclusive-OR indirect memory to	66-67	1	1

r_asm

BRANCHING				
ACALL addr11	Absolute call within current 2 K	11->F1	2	2
LCALL addr16	Long call to addr16	12	3	2
RET	Return from subroutine	22	1	2
RETI	Return from interrupt routine	32	1	2
AJMP addr11	Absolute jump within current 2 K	01->E1	2	2
LJMP addr16	Long jump unconditional	02	3	2
SJMP rel	Short jump to relative address	80	2	2
JC rel	Jump relative on carry = 1	40	2	2
JNC rel	Jump relative on carry = 0	50	2	2
JB bit,rel	Jump relative on direct bit = 1	20	3	2
JNB bit,rel	Jump relative on direct bit = 0	30	3	2
JBC bit,rel	Jump relative and clear on direct bit = 1	10	3	2

r_asm

A tener en cuenta:

- Tamaño
- Destino
- Opcode

r_asm

```
#undef _
#define _ (xc800_op)
#define _ARG(x) ARG, 0, x, buf
#define _ADDR11(x) ADDR11, ((x[1])+((x[0]>>5)<<8)), NULL, buf
#define _ADDR16(x) ADDR16, ((x[1])<<8)+((x[2])), NULL, buf
#define _OFFSET(x) OFFSET, ((x[1])), NULL, buf
#define _DIRECT(x) DIRECT, (x[1]), NULL, x
```

Notes on Program Addressing Modes:

addr16	-	Destination address for LCALL and LJMP may be anywhere within the 64 Kbytes of the active bank located in program space.
addr11	-	Destination address for ACALL and AJMP will be within the same 2-Kbyte page of program memory as the first byte of the following instruction.
rel	-	SJMP and all conditional jumps include an 8-bit offset byte. Range is + 127/– 128 bytes relative to the first byte of the following instruction.

r_asm

```
switch (op) {
case 0x10: return _{ "jbc bit,", 3, _ADDR16(buf) };
case 0x20: return _{ "jb bit,", 3, _ADDR16(buf) };
case 0x30: return _{ "jnb bit,", 3, _ADDR16(buf) };
case 0x40: return _{ "jc", 2, _OFFSET(buf) };
case 0x50: return _{ "jnc", 2, _OFFSET(buf) };
case 0x60: return _{ "jz", 2, _OFFSET(buf) };
case 0x70: return _{ "jnz", 2, _OFFSET(buf) };
case 0x80: return _{ "sjmp", 2, _OFFSET (buf) };

case 0x90: return _{ "mov dptr,", 3, _ADDR16(buf) }; // XXX
case 0xa0: return _{ "orl c, /bin", 2, NONE };
case 0xb0: return _{ "anl c, /bin", 2, NONE };

case 0xc0: return _{ "push", 2, _DIRECT (buf)};
case 0xd0: return _{ "pop", 2, _DIRECT (buf)};

case 0x02: return _{ "ljmp", 3, _ADDR16(buf) };
case 0x12: return _{ "lcall", 3, _ADDR16(buf) };
case 0x22: return _{ "ret", 1, NONE };
case 0x32: return _{ "reti", 1, NONE };
case 0x42: return _{ "orl direct, a", 2, _DIRECT (buf)};
case 0x92: return _{ "+, c;mov", 2, _DIRECT (buf) };
case 0xc2: return _{ "clr bit", 2, _DIRECT (buf) };
case 0xd2: return _{ "setb", 2, _DIRECT (buf) };
case 0xa2: return _{ "mov c,", 2, _DIRECT (buf) };
```

r_asm

```
0x00001100      a9d3            mov r1, 0xd3
0x00001102      6f              xrl a, r7
0x00001103      fa              mov r2, a
0x00001104      aad3            mov r2, 0xd3
0x00001106      67              xrl a, @r1
0x00001107      11ad            acall 0xad
0x00001109      d3              setb c
0x0000110a      2a              add a, r2
0x0000110b      122aea          lcall 0x2aea
0x0000110e      aed3            mov r6, 0xd3
0x00001110      a728            mov @r1, 0x28
0x00001112      0f              inc r7
0x00001113      4f              orl a, r7
0x00001114      11af            acall 0xaf
0x00001116      d3              setb c
0x00001117      0604            inc @r0
0x00001119      2a              add a, r2
0x0000111a      121305          lcall 0x1305
0x0000111d      20fa0d          jb bit, 0xfa0d
0x00001120      20f52a          jb bit, 0xf52a
```

r_anal

r_anal

anal_8051.c

```
RAnalPlugin r_anal_plugin_8051 = {
        .name = "8051",
        .arch = "8051",
        .bits = 8|16,
        .desc = "8051 CPU code analysis plugin",
        .license = "LGPL3",
        .op = &i8051_op,
        .set_reg_profile = set_reg_profile,
        .esil_init = esil_i8051_init,
        .esil_fini = esil_i8051_fini
};
```

r_anal

- op: Gestión de opcodes
- set_reg_profile: Definición de registros de la arquitectura
- esil_init:
- esil_fini: Emulación de esil

(esil = true)

- name: Nombre del plugin
- arch:
- bits:
- desc:
- license:

r_anal

```
static int set_reg_profile(RAnal *anal) {
        const char *p =
                "=PC    pc\n"
                "=SP    sp\n"
                "gpr    r0      .8      0       0\n"
                "gpr    r1      .8      1       0\n"
                "gpr    r2      .8      2       0\n"
                "gpr    r3      .8      3       0\n"
                "gpr    r4      .8      4       0\n"
                "gpr    r5      .8      5       0\n"
                "gpr    r6      .8      6       0\n"
                "gpr    r7      .8      7       0\n"
                "gpr    A       .8      8       0\n"
                "gpr    B       .8      9       0\n"
                "gpr    sp      .8      10      0\n"
                "gpr    pc      .16     12      0\n"
                "gpr    dptr    .16     14      0\n"
                "gpr    C       .1      16      0\n"
                "gpr    OV      .1      17      0\n";
        return r_reg_set_profile_string (anal->reg, p);
}
```

gpr = General Purpose Register

esil

“Evaluable Strings Intermediate Languaje” traduce a un lenguaje similar a Forth la semántica de cualquier opcode

“Evaluable Strings Intermediate Languaje” traduce a un lenguaje similar a Forth la semántica de cualquier opcode

esil

ESIL Opcode	Operands	Name	Operation
TRAP	src	Trap	Trap signal
$	src	Syscall	sysccall
$$	src	Instruction address	Get address of current instruction stack=instruction address
==	src,dst	Compare	v = dst - src ; update_eflags(v)
<	src,dst	Smaller	stack = (dst < src)
<=	src,dst	Smaller or Equal	stack = (dst <= src)
>	src,dst	Bigger	stack = (dst > src)
>=	src,dst	Bigger or Equal	stack = (dst > src)
<<	src,dst	Shift Left	stack = dst << src
>>	src,dst	Shift Right	stack = dst >> src
<<<	src,dst	Rotate Left	stack=dst ROL src
>>>	src,dst	Rotate Right	stack=dst ROR src
&	src,dst	AND	stack = dst & src
|	src,dst	OR	stack = dst | src
^	src,dst	XOR	stack = dst ^src
+	src,dst	ADD	stack = dst + src
-	src,dst	SUB	stack = dst - src
*	src,dst	MUL	stack = dst * src

ESIL Opcode	Operands	Name	Operation
/	src,dst	DIV	stack = dst / src
%	src,dst	MOD	stack = dst % src
!	src	NEG	stack = !!!src
++	src	INC	stack = src++
--	src	DEC	stack = src--
+=	src,reg	ADD eq	reg = reg + src
-=	src,reg	SUB eq	reg = reg - src
*=	src,reg	MUL eq	reg = reg * src
/=	src,reg	DIV eq	reg = reg / src
%=	src,reg	MOD eq	reg = reg % src
<<=	src,reg	Shift Left eq	reg = reg << src
>>=	src,reg	Shift Right eq	reg = reg << src
&=	src,reg	AND eq	reg = reg & src
|=	src,reg	OR eq	reg = reg | src
^=	src,reg	XOR eq	reg = reg ^ src
++=	reg	INC eq	reg = reg + 1
--=	reg	DEC eq	reg = reg - 1
!=	reg	NOT eq	reg = !reg
---	---	---	---
=[]	src,dst	poke	*dst=src
[]	src	peek	stack=*src

esil

sf,!,?{,0x1019,pc,=,}	if (!sf) pc =0x1019
eax,ebx,=	ebx=eax
eax,ebx,^,ebx	ebx=ebx ^ eax

0,sf,=,r_03,r_01,<,sf,=,0,zf,=,r_03,r_01,==,$z,zf,=,0,gf,=,r_03,r_01,>,gf,=	sf = 0; if(r_01 < r_03) sf = 1;
	zf = 0; if (r1 == r_3) zf = 1;
	gf = 0; if (r_01 > r_03) gf = 1;

esil

“Di 'qué' una vez más, te reto, te reto dos veces cabronazo, di 'qué' una vez más”

esil

```
[0x00000600]> pd
|||   0x00000600      d7              xchd a, r1 @r1
|||   0x00000601      cb              xch a, r3
|||   0x00000602      86fa            mov direct, @r0
|||   0x00000604      90d7cb          mov dptr, 0xd7cb
||    0x00000607      7f28            mov r7, 0x28
|||   0x00000609      0f              inc r7
|||   0x0000060a      0607            inc @r0
```

```
[0x00000700]> e asm.esil = true
[0x00000700]> s 0x600
[0x00000600]> pd
|||   0x00000600      d7              TODO,xchd a, r1 @r1
|||   0x00000601      cb              A,r3,A,=,r3,=
|||   0x00000602      86fa            r0,[1],0x10000,250,+,=[1],
|||   0x00000604      90d7cb          55243,dptr,=
|||   0x00000607      7f28            40,r7,=
|||   0x00000609      0f              r7,++=
|||   0x0000060a      0607            r0,++=[1],
```

Hex Value (max. 7fffffffffffff)	Decimal Value
28	40

r_anal

```
static int esil_xc800_init (RAnalEsil *esil) {
        if (esil->cb.user) {
                return true;
        }
        esil->cb.user = R_NEW0 (struct r_xc800_user);
        ocbs = esil->cb;
        esil->cb.hook_reg_read = xc800_hook_reg_read;
        esil->cb.hook_reg_write = xc800_hook_reg_write;
        xc800_is_init = true;
        return true;
}

static int esil_xc800_fini (RAnalEsil *esil) {
        if (!xc800_is_init) {
                return false;
        }
        esil->cb = ocbs;
        R_FREE (esil->cb.user);
        xc800_is_init = false;
        return true;
}
```

r_anal

```
#define PUSH1 "1,sp,+=,sp,=[1]"
#define POP1  "sp,[1],1,sp,-=,"
#define PUSH2 "1,sp,+=,sp,=[2],1,sp,+="
#define POP2  "1,sp,-=,sp,[2],1,sp,-=,"
```

```
case 0x40: /* jc   */ emitf("C,!,?{,%d,2,+,pc,+=,}", (st8)buf[1]); break;
case 0x50: /* jnc  */ emitf("C,""?{,%d,2,+,pc,+=,}", (st8)buf[1]); break;
case 0x60: /* jz   */ emitf("A,!,?{,%d,2,+,pc,+=,}", (st8)buf[1]); break;
case 0x70: /* jnz  */ emitf("A,""?{,%d,2,+,pc,+=,}", (st8)buf[1]); break;
```

r_debug

r_debug

debug_esil.c

```
RDebugPlugin r_debug_plugin_esil = {
        .name = "esil",
        .keepio = 1,
        .license = "LGPL3",
        .arch = "any", // TODO: exception!
        .bits = R_SYS_BITS_32 | R_SYS_BITS_64,
        .init = __esil_init,
        .step = __esil_step,
        .step_over = __esil_step_over,
        .cont = __esil_continue,
        .contsc = __esil_continue_syscall,
        .attach = &__esil_attach,
        .detach = &__esil_detach,
        .wait = &__esil_wait,
        .stop = __esil_stop,
        .kill = __esil_kill,
        .breakpoint = &__esil_breakpoint,
        .reg_profile = __esil_reg_profile,
        .reg_read = __reg_read,
};
```

Instalación de plugins

Instalación de plugins

- anal_xc800.c
- xc800.mk
- Makefile (Modificar)
- r_anal.h (Modificar)
- plugins.def.cfg (Modificar)

anal_xc800.mk

```
OBJ_xc800=anal_xc800.o
CFLAGS+=-I../asm/arch/xc800
STATIC_OBJ+=${OBJ_xc800}
#OBJ_xc800+=../../../../../../../../../../../../../../../../../../../../../../../../${LTOP}/asm/xc800.o
#OBJ_xc800+=${LTOP}/asm/arch/xc800/xc800.o
OBJ_xc800+=../../asm/arch/xc800/xc800.o
TARGET_xc800=anal_xc800.${EXT_SO}

ALL_TARGETS+=${TARGET_xc800}

 anal_xc800.${EXT_SO} ${OBJ_xc800}
```

anal_xc800.mk

```
include ../../config.mk
include ../../../mk/platform.mk

CFLAGS+=-I../../include -I../arch -Wall -shared $(PIC_CFLAGS) ${LDFLAGS_LIB} ${LDFLAGS_LINKPATH}..
CFLAGS+=-L../../util -lr_util -L../../anal -lr_anal -L../../reg -lr_reg
LDFLAGS+=${LINK}

CURDIR=

ifeq ($(WITHPIC),1)
all: ${ALL_TARGETS} ;

ALL_TARGETS=
# TODO: rename to enabled plugins
ARCHS=null.mk x86_udis.mk ppc_gnu.mk ppc_cs.mk arm_gnu.mk avr.mk xap.mk dalvik.mk sh.mk ebc.mk gb.mk
malbolge.mk ws.mk h8300.mk cr16.mk v850.mk msp430.mk sparc_gnu.mk sparc_cs.mk x86_cs.mk cris.mk 6502.mk
snes.mk riscv.mk vax.mk xtensa.mk rsp.mk xc800.mk
include $(ARCHS)

clean:
	-rm -f *.${EXT_SO} *.o ${STATIC_OBJ}

mrproper: clean
	-rm -f *.d ../arch/*/*/*.d

.PHONY: all clean mrproper
else
all clean mrproper:

.PHONY: all clean mrproper
endif
```

r_anal.h

```
...
...
extern RAnalPlugin r_anal_plugin_6502;
extern RAnalPlugin r_anal_plugin_snes;
extern RAnalPlugin r_anal_plugin_riscv;
extern RAnalPlugin r_anal_plugin_vax;
extern RAnalPlugin r_anal_plugin_i4004;
extern RAnalPlugin r_anal_plugin_xtensa;
extern RAnalPlugin r_anal_plugin_pic18c;
extern RAnalPlugin r_anal_plugin_rsp;
extern RAnalPlugin r_anal_plugin_xc800;
...
...
```

Fin

“¿Qué son 100 abogados en el fondo del mar?
Un buen comienzo”

Comunidad


sorry, noob question. Somebody knows which is the C function called when you overwrite some byte with w command?
20:43
Álvaro Felipe
Check cmd_write.c
20:43
That is the file where everything with 'w' is used
20:43
libr/core/cmd_write.c
20:44
thanks!
20:44

Referencias

https://github.com/radareorg/r2con/blob/master/2016/trainings/04-plugin-esil/slides.pdf

https://github.com/radare/radare2/wiki